FPGA-Based Acceleration of the 3D Finite-Difference Time-Domain Method

James P. Durbano
EM Photonics, Inc.
durbano@emphotonics.com

Fernando E. Ortiz
John R. Humphrey
Petersen F. Curt
Dennis W. Prather
University of Delaware
{ortiz, humphrey, curt, dprather}@ee.udel.edu

Abstract

In order to take advantage of the significant benefits afforded by computational electromagnetic techniques, such as the Finite-Difference Time-Domain (FDTD) method, solvers capable of analyzing realistic problems in a reasonable time frame are required. Although software-based solvers are frequently used, they are often too slow to be of practical use. To speed up computations, hardware-based implementations of the FDTD method have been recently proposed. In this paper, we present our most recent progress in the area of FPGA-based 3D FDTD accelerators. Three aspects of the design are discussed, including the host-PC interface, memory hierarchy, and computational datapath. Implementation and benchmarking results are also presented, demonstrating that this accelerator is capable of at least three-fold speedups over thirty-node PC clusters.

1. Introduction

Although the importance of fast, accurate computational electromagnetic (CEM) solvers is readily apparent, how to construct them is not. By nature, CEM algorithms are both computationally and memory intensive. Furthermore, the serial nature of most software-based implementations does not take advantage of the inherent parallelism found in many CEM algorithms. In an attempt to exploit parallelism, supercomputers and computer clusters are employed. However, these solutions can be prohibitively expensive and frequently impractical. Thus, a CEM accelerator would provide the community with much-needed processing power. This would enable iterative designs and permit the analysis of designs that would otherwise be impractical to analyze.

To this end, we are developing a full-3D, hardware-based accelerator for the Finite-Difference Time-Domain (FDTD) method [1]. This accelerator provides speedups of up to three orders of magnitude over single-PC solutions and surpasses the throughputs of PC clusters. In this paper, we briefly summarize the previous work in this area, where it has fallen short, and how our work fills the void. We then describe three of the major areas in the design: host-PC communication, the memory hierarchy, and the floating-point computational datapath. Next, implementation results are presented, along with detailed benchmarking statistics. These results demonstrate that our most recent 3D FDTD hardware accelerator is capable of three-fold speedups over thirty-node Beowulf clusters.

2. FDTD Method Background

In this section, we provide a very brief overview of the FDTD method for those unfamiliar with the algorithm. We refer to the reader to the citations throughout this paper for more in-depth discussions.

The FDTD method is a computational electromagnetic algorithm that can be applied to a wide array of problems, such as the coupled-ring oscillator shown in Figure 1, below. The algorithm must account for both the temporal and spatial dependences that exist in Maxwell's equations, which govern electromagnetic propagation. This method replaces the spatial derivatives found in Maxwell's equations with central-difference approximations, reducing the equations to simple operations such as addition, subtraction, and multiplication. The temporal derivatives are accounted for by "leapfrogging" the calculations of the electric and magnetic fields in time. The electric fields are computed first, followed by the magnetic fields. This process is repeated, permitting the simulation of electromagnetic propagation.

The design space to be modeled is discretized into a mesh of nodes. Each node has associated electric and magnetic field data, along with parameters describing the medium represented at that point. The electric and magnetic fields at each point are updated as described above. In a 3D FDTD formulation, as many as twelve

Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'04)
0-7695-2230-0/04 $ 20.00 IEEE

fields are required at each node, with each field update requiring a slightly different equation.

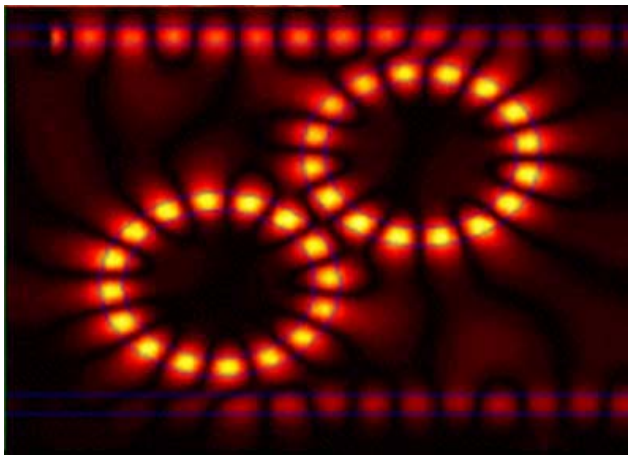

Figure 1. FDTD simulation This image was taken from EMPLab, a commercial electromagnetic simulation package from EM Photonics, Inc [2]. It shows the electromagnetic fields as they propagate through a coupled-ring oscillator. Our work aims to greatly increase the speed at which problems like this can be analyzed.

3. Related Work and Shortcomings

Several groups have attempted development of hardware-based FDTD solvers, dating back as far as the early 90s [3-13]. However, the physical implementation of a full-3D solver had never been described in the literature until our group published an architecture in early 2003 [6]. This solver represented a breakthrough in CEM acceleration, as it demonstrated that hardware-based 3D solvers were not only possible, but also capable of speedups that could surpass single PCs. The work in this paper represents the evolution of that architecture.

When analyzing our previous design, several weaknesses that impacted both the performance and scalability were uncovered. First, although the accelerator's performance could rival that of a single PC, scaling the design to support problems larger than approximately 80,000 nodes would require major modifications. Second, even if the proposed modifications outlined in [6] were implemented, the hardware would still only be able to rival small clusters. Although the power of a cluster in a desktop PC would be a tremendous achievement, a solution that could surpass small clusters and rival the throughputs of large (greater than 50 nodes) clusters was desired. Finally, the previous acceleration architecture did not support a true hardware source. For simplicity, the source was computed by the host PC each cycle and the value downloaded to the accelerator. Although this was sufficient to demonstrate the potential of hardware acceleration, a true, hardware-based source computation would eliminate this overhead.

The most recently reported progress in this field is presented in [3]. This paper outlines a pseudo-2D FDTD implementation. Although the authors demonstrate 24-fold speedups over single-PC implementations, the design exhibits several weaknesses. First, the implementation does not support a full-3D implementation, which is required for many electromagnetic simulations. Second, the implementation uses fixed-point, rather than floating-point, arithmetic components. Fixed-point representations sacrifice numeric precision to reduce area and increase speed. The authors present an error analysis to support the choice of fixed-point arithmetic, demonstrating that the precision loss does not severely impact the ultimate result. Unfortunately, this data is highly problem dependent. Third, the design does not support Perfectly Matched Layer (PML) boundary conditions, the most effective absorbing boundary conditions for the widest variety of problems [14]. Finally, due to the small amount of memory on the board, the design only supports relatively small problems. This limitation will be further compounded when extended to three dimensions because of the increased data storage requirements.

After analyzing the limitations of the architectures presented above, the goal was to design and implement the second generation (2G) of hardware-based CEM solvers. Once complete, the 2G accelerator would be able to solve large problems (greater than 100 million nodes), support PML boundaries and multiple source conditions, and perform at speeds rivaling those of large clusters.

4. Architecture Highlights

In this section, we describe the three main areas that comprise the hardware accelerator. These areas include the host-PC interface, memory hierarchy, and computational datapath. We begin with the host-PC interface.

4.1 Host-PC Interface

The acceleration platform consists of a host PC, Java CAD interface, and a custom, FPGA-based PCI card. The user describes the design to analyze by means of a CAD front end developed by EM Photonics, Inc [2]. This electromagnetic CAD software allows complex designs to be modeled and includes several toolboxes for specific applications, such as photonic bandgaps and diffractive optical elements. The front-end software then sends the problem-specific data, such as the mesh size, number of timesteps to execute, and source lookup table,

to the hardware via the PCI bus. The FDTD accelerator proceeds to update the fields, periodically sending the results back to the host computer for post-processing and visualization. The exact field values in the mesh to report back are determined by the user, who places detectors in the design. Detectors are communicated to the hardware once and the requested values are transferred back as they are computed.

If the user is only interested in steady-state values, the importance of a low-latency data transfer system is greatly reduced. For steady-state analysis, data transfers occur twice: initial data values are sent to the accelerator and the final, resultant data are sent back. However, many electromagnetic analyses require intermediate data values, and it is not uncommon for simulations to record data at several points and/or planes in the 3D mesh every *N* timesteps. In this case, it is important to optimize the transfer of updated field values back to the host PC to ensure that the communication does not slow down the computations. Otherwise, the speed increases obtained by the use of the accelerator may be offset by the data transfer overhead.

In the remainder of this section, two specific components of the PC/accelerator communication system are examined. Specifically, PCI bus controls, along with data buffering systems, are discussed. We begin with a discussion of the PCI controller.

4.1.1 PCI Control. Because the accelerator board relies on the host PC for initial problem parameters, post processing, and visualization, a means of communication must exist between the two. Both this architecture, and the architecture presented in [6] make use of the PCI bus for PC/accelerator communication. In the previous architecture, the PCI bus controller was implemented inside the FPGA. Unfortunately, designs that implement the controller inside the FPGA reduce logic resources that could otherwise be applied to the acceleration algorithm. In addition, the routing of the overall design becomes more complex, as the controller logic must be constrained and timed very precisely. For the 2G accelerator, PCI control is handled by an external ASIC. This chip, the PLX 9656, handles the intricate details of the PCI protocol, removing this burden from the FPGA [15].

The majority of data transfers occur via direct master reads and writes, with the accelerator card performing the role of "master." This means that the card initiates most data transfer requests, including the initial downloading of the problem parameters and the intermediate transfer of updated values back to the host PC.

Ultimately, the use of a custom chip simplified development, as it eliminated the need to handle the details of PCI-based exchanges, while at the same time freeing valuable FPGA resources.

4.1.2 Data Buffering. Simply placing the controller described above into the design will not necessarily result in the most efficient utilization of the PCI bus. As with many types of communication, the most efficient exchanges occur when transferring bursts of data. To this end, the accelerator contains a writeback buffering system. This system automatically buffers the necessary updated field values as directed by the user-defined detectors. When enough data has been buffered, a transfer is initiated to burst the entire buffer contents, allowing the board to utilize the PCI bus in the most efficient way possible. In addition, because these transfers are handled independently of the computational datapath, the communication latency does not slow down the computations.

In this section, two aspects of the PC/accelerator transfer system were described: the PCI controller and data buffering. These areas work together to allow efficient communication between the host and the accelerator, which is vital when large amounts of data need to be transferred between the two.

4.2 Memory Hierarchy

As outlined in [3, 6], the data storage and memory bandwidth requirements of the FDTD method are incredibly high, with representative problems (20 – 200 million nodes) requiring anywhere from 1 – 10 GB of RAM. Thus, there is a tremendous need for both large and fast data storage mechanisms in an FDTD accelerator. To this end, the 2G architecture makes use of large, fast main memories and smaller, faster internal cache memories. In this section, the memory subsystem is described. This system is composed of SRAM, DRAM, and internal FPGA BlockRAM. We now describe the use of DRAM and BlockRAM, with the discussion of SRAM postponed until Section 4.3.2 as it is a component of the source calculation system.

4.2.1 DRAM. The acceleration platform contains four, independent DDR SDRAM DIMMs, which were chosen for their high bandwidth, large storage capacity, wide-spread availability, easy upgrade path, and extensive design support. These channels have an aggregate bandwidth of 8.5 GB/s at 133 MHz, and are used to store information about the electromagnetic fields and material properties of the simulation.

Despite the large bandwidth provided by the DRAM channels, sporadic, random memory accesses still proved to be a tremendous design bottleneck. As mentioned in Section 4.1.2, bursting data to and from DRAM is the most efficient use of the memory channel. Thus, to take

full advantage of the bandwidth of the DDR modules, it is necessary to transfer large amounts of data between the DRAM and the FPGA. Because all of this information cannot immediately be inserted into the computational datapath, an internal buffering mechanism is required to temporarily store the data. This was achieved through the use of internal FPGA BlockRAM.

4.2.2 BlockRAM. The Virtex-II 8000 FPGA used in the design contains 3 Mb of internal BlockRAM. BlockRAM provides a dedicated, highly-configurable data store inside the FPGA. In the 2G design, BlockRAM is used as a dual-ported cache. Through careful organization, it is possible to burst large amounts of data from the external DDR modules and buffer them inside the FPGA in the BlockRAM. The dual-ported nature of the cache allows the design to maximize the communication bandwidth between the BlockRAM and computational datapath, keeping the computational pipelines full. The current caching system supports an aggregate bandwidth of up to 27 GB/s.

By coupling on-board SRAM and DRAM with internal BlockRAM cache blocks, a memory hierarchy capable of 36 GB/s was developed. This bandwidth helped satisfy the tremendous memory requirements of the FDTD method, allowing increased computational throughput.

In this section, we described the roles that DRAM and BlockRAM assume in the 2G architecture. Data are stored primarily in the larger DRAM modules until required for computations. They are then cached into the smaller, faster FPGA BlockRAM. As data leave the BlockRAM cache, they are fed into the computational datapath. In the next section, we describe several of the key components involved in this datapath.

4.3 Computational Datapath

In this section, we focus on the last major component of the FDTD accelerator: the computational datapath. We discuss the implementation of both the source-field computations and the field-update equations. A dataflow diagram picturing some of the components described in Sections 4.2 and 4.3 can be found in Figure 2, below. We begin with a discussion on why floating-point arithmetic units are used for computations.

4.3.1 Why Floating-Point Arithmetic. The FDTD accelerator described in this paper makes use of floating-point, rather than fixed-point, arithmetic. A 32-bit, IEEE 754-based floating-point representation was chosen after careful study for several reasons.

First, the performance penalty of using floating-point arithmetic units can be quite low if the units are carefully optimized. The performance of floating-point units in FPGAs has increased significantly in recent years with the arrival of such components as built-in multipliers and dedicated carry chains. By combining these resources with pipelining techniques, researchers have demonstrated floating-point units that operate in excess of 150 MHz [16]. Certainly, a fixed-point implementation requires less area and can be optimized to work at very high speeds, but floating-point units no longer present the gross bottleneck in computationally intensive designs that they once did.

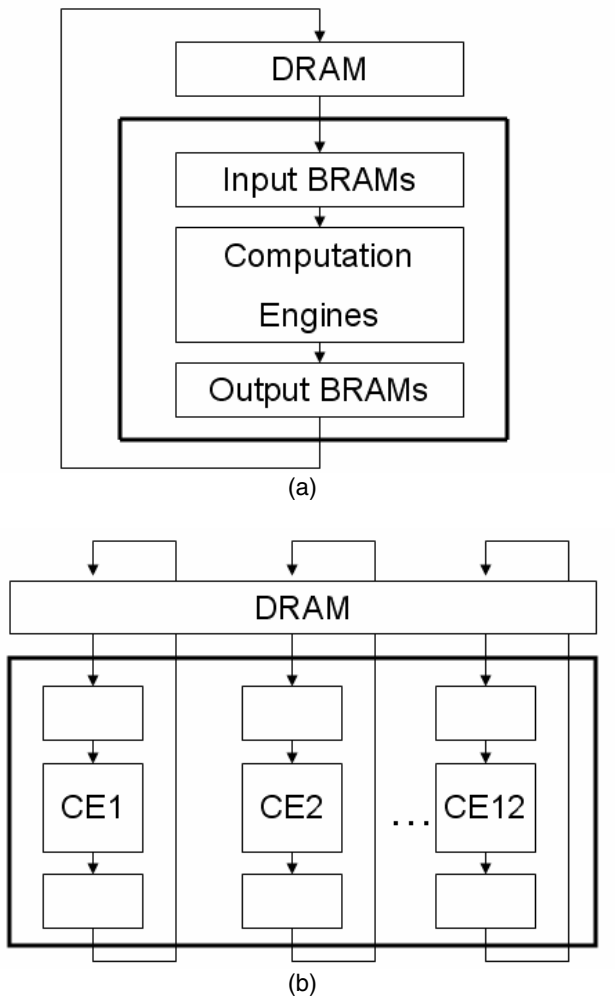

Figure 2. Dataflow architecture This diagram illustrates, at a very high level, the flow of data through the design. (a) This figure provides a top-level view of the DRAM/FPGA boundary (box) and shows the internal cache blocks in relation to the computation engines. (b) This figure contains the same information, but provides a more detailed view of the cache/computation engine configuration in the FPGA. Here we see that twelve computation engines are implemented, each one with an input and output cache block.

Second, the error introduced by fixed-point quantization increases over time. In time-iterative algorithms, such as FDTD, error that is introduced accumulates as it propagates, growing exponentially over time. Even if the differences in the final results are small enough to be imperceptible to the eye, steady state analysis of the results can greatly increase the error. This is because a small error in the time domain can have a tremendous impact on the error when converted to the frequency domain.

Third, in order to analyze a variety of problems, it is necessary to support a wide range of problem parameters. These parameters, such as material-based coefficients, have values that can easily vary several orders of magnitude. In a fixed-point arithmetic system, this large dynamic range will result in reduced precision. However, it is critical that the acceleration architecture support a wide variety of materials without impacting the accuracy of the system. Furthermore, a fixed-point implementation may affect the ability of the system to successfully implement the PML boundary conditions. The nature of the PML absorbing boundary conditions requires that a large range of material coefficient values be represented simultaneously. Thus, choosing a fixed-point representation to model both fields and widely varying material parameters could quickly prove problematic, leading to a rapidly growing loss of precision over time.

Finally, the differences in the relative magnitudes of the electric and magnetic fields result in an immediate loss of precision. For example, when considering the simple case of a wave propagating in free space, there is a difference of at least two orders of magnitude between the amplitudes of the electric and magnetic fields. For fixed-point implementations, both the larger electric fields and smaller magnetic fields must be represented using the same number of integer bits, leaving fewer fractional bits available for precision.

For these reasons, an FDTD accelerator that utilizes floating-point, rather than fixed-point, arithmetic is desired.

4.3.2 Source Computation. In electromagnetic simulations, there must be an excitation source. This source is responsible for initiating the electromagnetic waves that are then propagated through and/or scattered by the design under analysis. Sources can take on many forms, including point sources that represent a single point of electromagnetic energy, such as a dipole antenna, and plane waves that represent the fields as seen a great distance from a point source, such as those used by radar.

The introduction of sources into FDTD algorithms can take on many different forms, including total/scattered field formulations and hard/soft sources [17]. Thus, a hardware accelerator needs to be able to support several types of sources, each with a unique simulation method. To accommodate this, the acceleration platform makes use of a source lookup table and interpolation techniques [18]. These techniques will be more fully described in a future paper, but are presented here at a very high level to provide a basic understanding.

Prior to downloading the initial problem data, the front-end software pre-computes several parameters. Among these is the value of the source at many different times and locations. Based on the user-selected source, a lookup table is generated. This table, which is partially stored in both the SRAM and FPGA BlockRAM, is indexed by a parameter that is based on the current timestep and the relative location of the node in the mesh. From this information, a given row in the LUT is fetched, and the source field data are interpolated to best approximate values that are not explicitly listed in the table.

One of the primary reasons a LUT was chosen was that it provides a great deal of flexibility. Because of the variety of sources that exist, a solution that can accommodate many sources is needed. Implementing computational hardware to handle point sources and plane waves, along with any type of enveloping function, including temporal and spatial windows, would require a tremendous amount of hardware and could easily require hundreds of cycles to compute. By using a LUT, computation of the source is achieved by simply fetching the pre-computed source values and then generating a more accurate value by means of a linear interpolation.

In this section, we briefly described how electromagnetic sources are introduced into the acceleration hardware. In the next section, we describe the most computationally intensive portion of the design: the implementation of the field-update equations.

4.3.3 Computation Engines. The computation engines implement the field-update equations, which represent a discretized form of Maxwell's equations. These equations are responsible for propagating the electromagnetic fields and are at the heart of any hardware-based implementation of the FDTD method.

The computation engines evaluate equations of the form:

$$UpdatedField = A * PreviousField + B * IncidentField1 + C * IncidentField2 + D * (Field1 - Field2 + Field3 - Field4),$$

where *PreviousField* represents the value of the field updated during the previous timestep, *IncidentField1* and *IncidentField2* represent parameters provided by the source LUT, *Field1-4* represent surrounding

electromagnetic field values, and *A-D* are problem-dependent coefficients, which are based on aspects such as the medium of propagation.

As outlined in [3, 6], there is a great deal of parallelism in the FDTD method that can be exploited by computational hardware. In the current architecture, twelve equations of the form shown above are evaluated in parallel. This allows twelve fields to be updated simultaneously. Although the FDTD method could easily support more parallelism, the resource requirements would be too great to implement such an architecture in current FPGA technology.

In this section, three areas of the acceleration architecture were highlighted. Specifically, the host-PC interface, memory hierarchy, and computational datapaths were discussed. Each general area was then subdivided into more specific areas, such as the use of internal FPGA memory as a cache and the implementation of the field-update equations. A discussion on why floating-point arithmetic is used was also presented. In the next section, we describe the implementation results and performance of the system. Included are benchmarking data of other FDTD solvers, such as single PCs and computer clusters.

5. Implementation and Performance

The custom-designed accelerator board supports up to 16 GB of DDR SDRAM, 36 Mb of DDR SRAM, a Xilinx Virtex-II 8000 FPGA, and a PLX 9656 external PCI controller (see Figures 3 and 4, below). This card fits in standard PCs and supports the newer, PCI 64/66 bus for efficient PC/accelerator communication. These components can provide a peak external memory bandwidth of 9.7 GB/s.

The acceleration architecture was coded in VHDL and has passed functional simulations. Currently, a few minor timing issues preventing a complete hardware analysis are being resolved. After correct operation has been verified, optimization of the design will begin. Several simplifications, mostly related to the timing of the design, were implemented so that correct operation could be demonstrated. However, these aspects will be optimized to increase the throughput of the system. The design supports the Perfectly Matched Layer absorbing boundary conditions and perfect electric conductor walls. It also supports both point and plane-wave sources, including variations such as Gaussian-modulated sources, and is highly scalable to support others

The 2G system is expected to update approximately 30 million nodes per second (Mnps). To put this number in context, a 3.0 GHz Dell workstation with 2 GB PC 3200 DDR SDRAM can perform up to 4.3 Mnps (when the problem fits inside the cache), but averages around 1.3 Mnps. However, these throughputs are achieved only for relatively small problems. As problem sizes get large and surpass the physical RAM in the PC, hard disk swapping begins and throughputs drop to as low as 0.08 Mnps. Thus, our solution, which guarantees a constant 30 Mnps, regardless of problem size, represents up to three orders of magnitude improvement over existing solutions (see Figure 5, below). It is important to note that the software benchmarks are based on standard FDTD solvers, optimized to run on a PC. The code does not model the hardware implementation, as this would most likely lower the throughput of the PC and provide skewed performance benchmarks, which benefit the hardware accelerator.

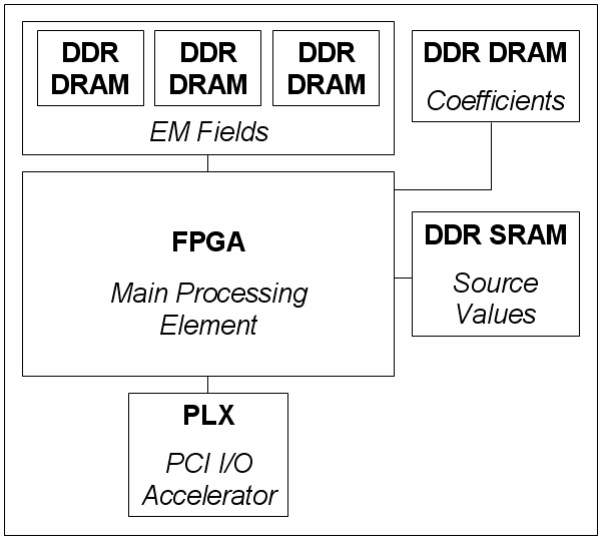

Figure 3. Custom board block diagram Pictured is a high-level block diagram of the custom board. It clearly shows the external SRAM, DRAM, and PLX components described in Section 4.

Figure 4. The custom board This is a picture of the custom, FPGA-based board that holds the acceleration architecture. It includes the largest FPGA on the market and supports up to 16 GB DDR SDRAM.

Another way to understand the Mnps measurement is to look at the number of floating-point units required to update a node. In the current design, a total of 96 floating-point adders and multipliers are required to

implement the basic equations. Another 48 floating-point units are necessary to compute the electromagnetic source. Thus, to update a single node requires the use of 144 floating-point units. Because of the pipelined design, all of these units can be performing at the same time. With the current clock rate, this results in a peak computational throughput of almost 15 GFLOPS.

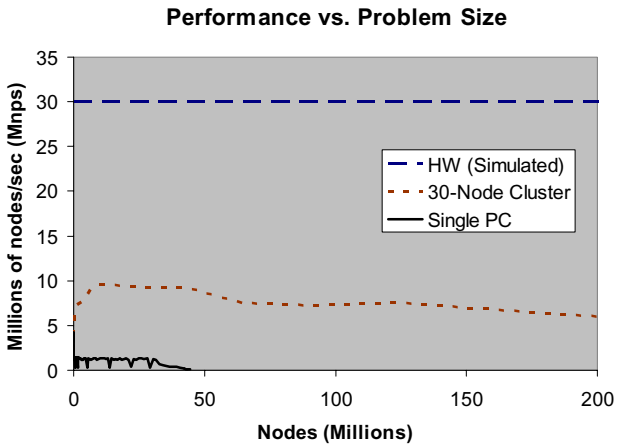

Figure 5. Benchmarking results This graph compares three FDTD analysis systems: a single PC, a 30-node cluster, and the acceleration hardware. It clearly shows the tremendous speedup provided by the acceleration hardware over standard, FDTD solver solutions. The cluster is composed of 30 machines identical to the "Single PC."

The expected throughput of the system was determined by combining how fast the design was to be clocked (a reasonable 100 MHz) and the number of pipeline stages required to complete the update of a node. Various overheads, such as memory accesses, were then studied. However, these overheads proved minimal as much of the data transfer latency is hidden by caching data and overlapping computations with memory accesses. Functional simulations of the design also support this estimate.

In terms of size, the 2G accelerator can support problems with up to 256 million nodes. The Dell workstation described above can only support problems with up to approximately 44 million nodes, as larger simulations require more memory than can be allocated by the virtual memory system. This limitation is imposed by 32-bit, PC-based operating systems, which support a maximum program address space of 3 GB.

The design was implemented using ActiveHDL 6.1 for simulation, Synplify Pro 7.5 for synthesis, and Xilinx ISE 6.1 for place-and-route. Post-synthesis reports indicate the following FPGA resource utilization: 94% of the 18x18 integer multipliers, 95% of the BlockRAM, 87% of the LUTs, and 61% of the flip-flops. In the near future, the design will be modified to use more built-in multipliers, reducing the required number of LUTs and increasing speed. Because automatic place and route tools frequently perform poorly on very large designs, manual floorplanning is required.

Finally, it is important to compare the relative costs of the accelerator with those of other electromagnetic solvers, such as the single-PC and Beowulf-cluster solutions presented in this section. The accelerator cards can be manufactured for less than $10,000 (US). Although the accelerator cards cost more than single-PC solutions, this additional cost brings with it a 23-fold speedup and an almost 6-fold increase in the maximum problem size. When compared with 90-node PC clusters, although the throughputs are approximately the same, the cost of the accelerator is much less. Furthermore, our solution requires much less area and power, making it even more attractive.

In this section, the actual implementation of the accelerator was presented. The performance of the hardware was then compared to that of single PCs and thirty-node clusters. The hardware accelerator surpassed the performance of both, proving that hardware-based acceleration of the FDTD method will soon be a viable option for the electromagnetics community.

6. Conclusion

This paper began with a look at the previous research in this field, highlighting both the successes (and weaknesses) of these designs. A new architecture was then described, focusing on three main areas: host-PC communication, the memory hierarchy, and the computational datapath. Results were then presented, which compared this new architecture with existing FDTD analysis platforms, including single PCs and Beowulf clusters.

To date, the system has been fully designed and simulated using VHDL. A custom, FPGA-based PCI card has also been manufactured. The acceleration hardware can accommodate larger problems than PCs and can solve them orders of magnitude faster. This system represents a new era in computational electromagnetic simulations, greatly increasing the speed at which current problems can be solved and permitting the analysis of new designs.

References

[1] K. S. Yee, "Numerical solution of initial boundary value problems involving Maxwell's equations in isotropic media," *IEEE Transactions on Antennas and Propagation*, vol. 14, pp. 302-307, 1966.

[2] "EM Photonics, Inc.," www.emphotonics.com.

[3] W. Chen, P. Kosmas, M. Leeser, and C. Rappaport, "An FPGA implementation of the two-dimensional finite-difference time-domain (FDTD) algorithm," presented at the *Twelfth ACM International Symposium on Field-Programmable Gate Arrays* (FPGA), 2004.

[4] J. P. Durbano, F. E. Ortiz, J. R. Humphrey, D. W. Prather, and M. S. Mirotznik, "Implementation of three-dimensional FPGA-based FDTD solvers: an architectural overview," presented at the *11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines* (FCCM), 2003.

[5] J. P. Durbano, F. E. Ortiz, J. R. Humphrey, D. W. Prather, and M. S. Mirotznik, "Development of hardware-based FDTD solvers: an analysis of design strategies and implementation results," presented at the *7th World Multi-Conference on Systemics, Cybernetics, and Informatics* (SCI), 2003.

[6] J. P. Durbano, F. E. Ortiz, J. R. Humphrey, D. W. Prather, and M. S. Mirotznik, "Hardware implementation of a three-dimensional finite-difference time-domain algorithm," *IEEE Antennas and Wireless Propagation Letters*, vol. 2, pp. 54-57, 2003.

[7] J. P. Durbano, "Hardware implementation of a 1-dimensional finite-difference time-domain algorithm for the analysis of electromagnetic propagation," University of Delaware, Newark, M.E.E. Thesis, 2002.

[8] P. Placidi, L. Verducci, G. Matrella, L. Roselli, and P. Ciampolini, "A custom VLSI architecture for the solution of FDTD equations," *IEICE Transactions on Electronics*, vol. E85-C, pp. 572-577, 2002.

[9] K. Ramachandran, "Unstructured finite element computations on configurable computers," Virginia Polytechnic Institutue and State University, Blacksburg, VA, USA, MS Thesis, 1998.

[10] R. N. Schneider, L. E. Turner, and M. M. Okoniewski, "Application of FPGA technology to accelerate the finite-difference time-domain (FDTD) method," presented at the *Tenth ACM International Symposium on Field-Programmable Gate Arrays* (FPGA), Monterey, CA, 2002.

[11] L. Verducci, P. Placidi, G. Matrella, L. Roselli, F. Alimenti, P. Ciampolini, and A. Scorzoni, "A feasibility study about a custom hardware implementation of the FDTD algorithm," presented at the *27th General Assembly of the URSI*, Maastricht, Netherlands, 2002.

[12] J. R. Marek, "An investigation of a design for a finite-difference time domain (FDTD) hardware accelerator," Air Force Institute of Technology, Wright-Patterson AFB, MS Thesis, 1991.

[13] J. R. Marek, M. A. Mehalic, and J. Andrew J. Terzuoli, "A dedicated VLSI architecture for finite-difference time domain calculations," presented at the *8th Annual Review of Progress in Applied Computational Electromagnetics*, Naval Postgraduate School, Monterey, CA, 1992.

[14] J.-P. Berenger, "Three-Dimensional Perfectly Matched Layer for the Absorption of Electromagnetic Waves," *Journal of Computational Physics*, vol. 127, pp. 363-379, 1996.

[15] "PLX Technology, Inc.," www.plxtech.com.

[16] F. E. Ortiz, "Design of FPGA-oriented floating-point adders and multipliers in VHDL," University of Delaware, Newark, M.E.E. Thesis, 2003.

[17] A. Taflove and S. C. Hagness, *Computational Electrodynamics: The Finite-Difference Time-Domain Method*, 2nd ed. Norwood: Artech House, 2000.

[18] F. E. Ortiz, J. R. Humphrey, J. P. Durbano, and D. W. Prather, "A Study on the Design of Floating-Point Functions in FPGAs," presented at the *International Conference on Field Programmable Logic and Applications* (FPL), 2003.